PROBLEM. Music Recommendation [Score 50/50] (contributed by A. Caronongan)

I. Introduction

For this problem, you are going to create a module of a music recommendation system given a list of songs that have been played by the user. More specifically, you are going to perform the following:

1.) Identify the user's favorite music
2.) Recommend to the user a list of music they would probably like based on the artist of their favorite music

To accomplish this, you will need to properly traverse through different arrays of structures that represent the artists, music, and the played music indicated in the collection. More specifically, you will be provided with the following:

1.) An array of structures representing the artists of songs/music that exist in the collection
2.) An array of structures representing the songs/music available in the music library
3.) An array of structures representing the list of songs/music that have been played by the user

II. Artist structure

An artist attributed to songs that exist in the music library is represented with the following structure.

```
struct artistTag
{
      int nArtistID;
     string strArtistName;
};
```

Where each field corresponds to the following:

`nArtistID` – An integer representing an artist's unique ID. This value is considered **unique** for each entry, and no two artists within the collection will contain similar IDs. Likewise, the nArtistID **MAY NOT NECESSARILY** match the index within the array of artists (**See Section I**) provided within the music library.

`strArtistName` – Consists of the artist's name represented as a string. Contains at most 100 characters and may contain whitespace characters (' ' or spaces) but not newline characters ('\n').

Refer to Table 1 for a sample representation of artists within the music library's collection. Do take note that Table 1 does not fully represent the list of artists available within the collection.

nArtistID	1	2	3	4	5	...
strArtistName	"Blackpink"	"Dua Lipa"	"Hatsune Miku"	"BTS"	"LANY"	...

Table 1. Sample representation of artists. Each nArtistID is considered a unique value.

III. Song structure

Each song in the music library is represented with the following structure:

```
struct songTag
{
      int nSongID;
     string strTitle;
     int nArtistID;
    int nPlayDuration;
};
```

Where each field corresponds to the following:

nSongID – An integer representing a song's unique ID. This value is considered **unique** for each entry, and no two songs within the collection will contain similar IDs (this means that if a song has an ID that is the value of 3, no other song in the array will contain an ID with a value of 3). Take note that the nSongID of a particular song **MAY NOT NECESSARILY** match its corresponding index within the array of songs (**See Section I**) provided representing the music library.

strTitle – Consists of the song's title represented as a string. Contains at most 100 characters and may contain whitespace characters (' ' or spaces) but not newline characters ('\n').

nArtistID – An integer representing the ID of the associated artist of that song. Refer to **Section II (Artist Structure)** for more information about this.

nPlayDuration – An integer representing the total time (in seconds) the song entry has been played within the player. Upon starting the program, all song entries will have this value set to 0, with the corresponding final value to be determined by the play log indicated in the next section (**See Section IV.**)

Refer to Table 2 for a sample representation of artists within the music library's collection. Do note that the table does not fully represent the complete list of songs within the collection.

nSongID	1	2	3	4	5	...
strTitle	"Dynamite"	"As if it's your last"	"IDGAF"	"Lovesick Girls"	"Tale of the Deep Sea Lily"	...
nArtistID	4	1	2	1	3	...
nPlayDuration	0	0	0	0	0	...

Table 1. Sample representation of songs in the library. The nArtistID, as indicated in Section II, indicates the artist associated with each song entry. This means that the song "Dynamite" is associated with the artist with the nArtistID value of 4, meaning that the artist of "Dynamite" is "BTS". Likewise, the artist of "IDGAF" (which has an nArtistID of 2) is "Dua Lipa" (matching ID no. of 2).

IV. Played Song Structure

(NOTE: The previous structure indicated in section III refers to songs that exist in the library. The following structure refers to each song that has been played within the music player)

Each played song is represented with the following structure:

```
struct playedTag
{
        int nSongID;
       int nDuration;
};
```

Where each of the following fields correspond to the following:

nSongID - Corresponds to the nSongID of the corresponding song that exists within the music library (Refer to **Section III (Song Structure)** for more information about this).

nDuration - Corresponds to the time in seconds that the song has been played before switching to another song. It is also observed that a very high nDuration value indicates that the song has been put in repeat.

Do note that entries in the play log structure MAY contain duplicate entries with the same nSongID. This indicates that the particular song has been played several times. So if there exists two entries with an nSongID = 4 ("Lovesick Girls") indicating play times of 80 and 150 in the nDuration, then the total play time of "Lovesick Girls" is 230 respectively.

V. Specific Objectives

Going back to the objectives indicated in Section I, you are going to identify the user's favorite songs and favorite artists. More specifically, you are going to complete the following functions:

NOTE: For all 3 functions, refer to the provided skeleton codes for their respective function prototypes and parameters

1.) **`printSongInformation()`** – You are to output the song's information, indicating the title and the artist of a given song's ID and the base address of the array of structures representing the list of songs and artists respectively.

2.) **`getMostPlayed()`** – Given the base address of the array of structures representing the collection of songs that have been played, this function returns the `nSongID` of the song which has been played for the longest time. You may assume that no two songs will end up being the most played songs, and only one song will end up being the user's "favorite" song. Likewise, do not assume that all music files in the collection will be played at least once. It's possible that there may be some music files that will be left unplayed. A song's final total playtime will be determined by their corresponding `nPlayDuration` field.

3.) **`recommendMusic()`** – This function will print out ALL songs within the collection that matches the artist associated with the user's favorite song, by receiving a corresponding `nFavoriteSongID` passed as the integer parameter. The function prints out all songs by the artist of the song with the corresponding `nFavoriteSongID` **EXCEPT** the song with the corresponding `nSongID` matching the provided ID parameter (first parameter). This function displays the recommended songs in the order they appear within the collection (based on their index within the array, and not their `nSongID`), with entries having a lower index within the array of songs being displayed first. (e.g. If song A appears in index 0, and song B appears at index 2, print out song A first).

   Ex: If the user's favorite song is is "Lovesick Girls" (nSongID == 4), and the corresponding artist of that song has an nArtistID of 1 ("Blackpink"), print out ALL other songs where the nArtistID == 1 (Blackpink) EXCEPT the song with nSongID == 4 ("Lovesick Girls").

VI. General Instructions and Restrictions

1. Use and follow the given file templates/skeleton files.
2. Your solution should NOT include any input or output statements (e.g., scanf() or printf())
3. You are only allowed to use strlen(), strcmp(), strcat(), and strcpy() from string.h. No other functions from string.h can be used.
4. You are not allowed to include other header files in the templates, other than

those indicated.

5. Your submitted file LASTNAME-music.c **should not** contain main().

You are given the following files for this problem:

(1) `music.h` which contains the type declaration and function prototypes that will be used in the program. This file cannot be modified.

(2) `LASTNAME-Music.c` which contains some initial code that you'll need to complete. Comments indicate the requirement and restrictions imposed for the solution. For this file, the student's implementation should not include any scanf) or added printf() statements. **The only printf statements you are allowed to modify are the printf statements in lines 78, 79, 140, and 141 respectively**. This file should not be modified to contain the main(), and likewise .

(3) `musicMain.c` which contains the main(). This main function first prints out the list of songs in the library and can be used to check your function implementations in LASTNAME-Music.c respectively. This file should not be modified.

Note: 4 test cases have already been added for you to check your program. Upon running musicMain.c, you will be prompted to enter a test case ID between 1-4 inclusively to check your program.

(4) `EXPECTED OUTPUTS` is a folder containing 5 text files, namely the files named "EXPECTED SONG LISTS" that should pertain to the list of songs loaded in the database upon starting musicMain.exe and the expected outputs of the four test cases named "EXPECTED OUTPUT - Test Case no. N.txt"

(5) `.music files` pertaining to `artists.music`, `songs.music`, and `playLog1 - 4.music`. These files should be in the same directory as your `musicMain.c` and `LASTNAME-Music.c` file(s) and **SHOULD NOT** be touched at any point.

DELIVERABLE: Your C program source file `LASTNAME-Music.c` with your own last name as filename. For example, if your lastname is SANTOS, then the source file should be named as `SANTOS-Music.c`. Upload your source file in AnimoSpace before the indicated submission time.

TESTING & SCORING:

- Your program will be compiled via gcc -Wall, using C99 standard. Thus, for each function that does not compile successfully, the score for that function is 0.
- Your program will be tested by your instructor with other TEXT FILES (contents are different from the ones given to you) and with function calls of different parameter values.
- Full credit will be given for the function only if the student's implementations are all correct for all the test values used by the instructor during checking AND only if the student's implementation complied with the requirement and did not violate restrictions. Deductions will be given if not all test cases produce correct results OR if restrictions were not followed.